Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №7  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Обход графа в глубину»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Обход графа в глубину

**Лабораторное задание**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.

3. \* Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

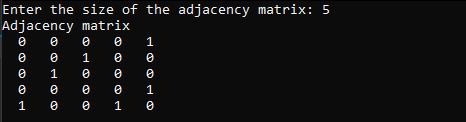
**Задание 2 \***

1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Ход работы**

**Задание 1**

1. Сгенерировал (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Вывел матрицу на экран.



1. Для сгенерированного графа осуществил процедуру обхода в глубину.



3. \* Реализовал процедуру обхода в глубину для графа, представленного списками смежности.

****

****

**Задание 2 \***

1. Для матричной формы представления графов выполнил преобразование рекурсивной реализации обхода графа к не рекурсивной.



**Листинг**

* Header.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

void fill\_arr(int\*\* Arr, int size);

void print\_arr(int\*\* Arr, int size);

struct node\* createNode(int v);

struct Graph\* createGraph(int vertices);

void addEdge(struct Graph\* graph, int src, int dest);

void printGraph(struct Graph\* graph);

struct Graph\* transformGraph(int\*\* Arr, int size);

void DFS(int\*\* Matrix, int size);

void Dfs(int\*\* Matrix, int size, int vertex, int\* visited);

void DFSList(struct Graph\* graph, int size);

void DfsList(struct Graph\* graph, int vertex, int\* visited);

void iterativeDFS(int\*\* Matrix, int size);

void iterativeDfs(int\*\* Matrix, int size, int vertex, int\* visited);

struct nodeStack\* create\_node(int inf);

int pop();

void push(int inf);

int stackEmpty();

* Stack.cpp

#include "Header.h"

struct nodeStack\* head = NULL;

struct nodeStack

{

int inf;

struct nodeStack\* next;

};

void push(int inf)

{

struct nodeStack\* p = NULL;

p = create\_node(inf);

if (head == NULL && p != NULL)

{

head = p;

}

else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

return;

}

int pop()

{

if (head == NULL)

{

return -1;

}

int data = 0;

struct nodeStack\* struc = head;

head = struc->next;

data = struc->inf;

free(struc);

return data;

}

int stackEmpty()

{

if (head == NULL)

return 1;

else

return 0;

}

struct nodeStack\* create\_node(int inf)

{

struct nodeStack\* p = NULL;

int ext = 0;

if ((p = (nodeStack\*)malloc(sizeof(struct nodeStack))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

p->inf = inf;

p->next = NULL;

return p;

}

* Lab7.cpp

#include "Header.h"

int main(void)

{

srand(time(NULL));

int sizeMatrix;

int\*\* Arr;

printf("Enter the size of the adjacency matrix: ");

scanf("%d", &sizeMatrix);

Arr = (int\*\*)malloc(sizeMatrix \* sizeof(int\*));

for (int i = 0; i < sizeMatrix; ++i)

{

Arr[i] = (int\*)malloc(sizeMatrix \* sizeof(int));

}

printf("Adjacency matrix\n");

fill\_arr(Arr, sizeMatrix);

print\_arr(Arr, sizeMatrix);

struct Graph\* graph = transformGraph(Arr, sizeMatrix);

printGraph(graph);

printf("\nDFS in the adjacency matrix");

DFS(Arr, sizeMatrix);

printf("\n\nDFS in the adjacency list");

DFSList(graph, sizeMatrix);

printf("\n\nIterative DFS in the adjacency matrix");

iterativeDFS(Arr, sizeMatrix);

free(Arr);

return(0);

}

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjLists;

};

void DFS(int\*\* Matrix, int size)

{

int\* visited = (int\*)calloc(size, size \* sizeof(int));

for (int i = 0; i < size; i++)

{

if (visited[i] == 0) {

printf("\n");

Dfs(Matrix, size, i, visited);

}

}

}

void Dfs(int \*\*Matrix, int size, int vertex, int\* visited)

{

visited[vertex] = 1;

printf("%d ", vertex);

for (int i = 0; i < size; i++)

{

if (Matrix[vertex][i] == 1 && visited[i] == 0)

{

Dfs(Matrix, size, i, visited);

}

}

}

void DFSList(struct Graph\* graph, int size)

{

int\* visited = (int\*)calloc(size, size \* sizeof(int));

for (int i = 0; i < size; i++)

{

if (visited[i] == 0) {

printf("\n");

DfsList(graph, i, visited);

}

}

}

void DfsList(struct Graph\* graph, int vertex, int\* visited)

{

visited[vertex] = 1;

printf("%d ", vertex);

struct node\* temp = graph->adjLists[vertex];

while (temp)

{

int i = temp->vertex;

if (visited[i] == 0)

DfsList(graph, i, visited);

temp = temp->next;

}

}

void iterativeDFS(int\*\* Matrix, int size)

{

int\* visited = (int\*)calloc(size, size \* sizeof(int));

for (int i = 0; i < size; i++)

{

if (visited[i] == 0) {

printf("\n");

iterativeDfs(Matrix, size, i, visited);

}

}

}

void iterativeDfs(int\*\* Matrix, int size, int vertex, int\* visited)

{

push(vertex);

while (!stackEmpty())

{

vertex = pop();

if (visited[vertex] == 1)

continue;

visited[vertex] = 1;

printf("%d ", vertex);

for (int i = 0; i < size; i++)

{

if (Matrix[vertex][i] == 1 && visited[i] == 0)

{

push(i);

}

}

}

}

void print\_arr(int\*\* Arr, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

printf("%3d ", Arr[i][j]);

}

printf("\n");

}

}

void fill\_arr(int\*\* Arr, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

if (i == j)

{

Arr[i][j] = 0;

continue;

}

if (rand() % 3 == 1)

{

Arr[i][j] = 1;

Arr[j][i] = 1;

}

else

{

Arr[i][j] = 0;

Arr[j][i] = 0;

}

}

}

}

struct node\* createNode(int v)

{

struct node\* newNode = (struct node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = (struct node\*\*)malloc(vertices \* sizeof(struct node\*));

for (int i = 0; i < vertices; i++)

graph->adjLists[i] = NULL;

return graph;

}

struct Graph\* transformGraph(int\*\* Arr, int size)

{

struct Graph\* graph = createGraph(size);

for (int i = 0; i < size; i++)

{

for (int j = i; j < size; j++)

{

if (Arr[i][j] == 1)

{

addEdge(graph, i, j);

}

}

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph)

{

printf("\nAdjancecy list");

for (int v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\n|%d|", v);

while (temp)

{

printf(" -> %d", temp->vertex);

temp = temp->next;

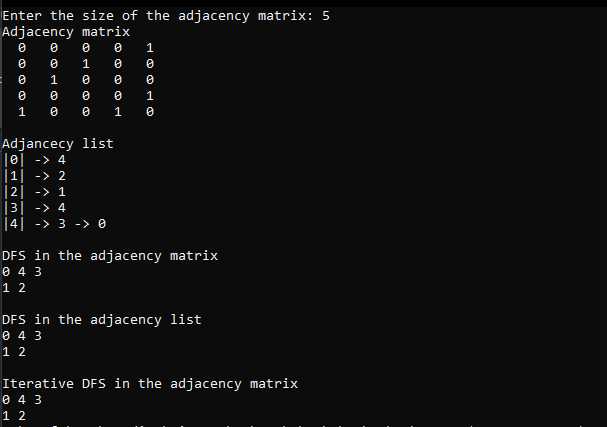
}

}

printf("\n");

}

**Результат работы программы**

****

**Вывод**

В результате выполнения лабораторной работы я научился выполнять обход графа в глубину.